Foundation of Cryptography, Lecture 8
Secure Multiparty Computation

Iftach Haitner, Tel Aviv University

Tel Aviv University.

May 20, 2014

Section 1

The Model

Multiparty Computation

- Multiparty Computation – computing a functionality f

Multiparty Computation

- Multiparty Computation – computing a functionality f
- Secure Multiparty Computation: compute f in a “secure manner"

Multiparty Computation

- Multiparty Computation – computing a functionality f
- Secure Multiparty Computation: compute f in a “secure manner"
  - Correctness

Multiparty Computation

- Multiparty Computation – computing a functionality f
- Secure Multiparty Computation: compute f in a “secure manner"
  - Correctness
  - Privacy

Multiparty Computation

- Multiparty Computation – computing a functionality f
- Secure Multiparty Computation: compute f in a “secure manner"
  - Correctness
  - Privacy
  - Independence of inputs

Multiparty Computation

- Multiparty Computation – computing a functionality f
- Secure Multiparty Computation: compute f in a “secure manner"
  - Correctness
  - Privacy
  - Independence of inputs
  - Guaranteed output delivery

Multiparty Computation

- Multiparty Computation – computing a functionality f
- Secure Multiparty Computation: compute f in a “secure manner"
  - Correctness
  - Privacy
  - Independence of inputs
  - Guaranteed output delivery
  - Fairness : corrupted parties should get their output iff the honest parties do

Multiparty Computation

- Multiparty Computation – computing a functionality f
- Secure Multiparty Computation: compute f in a “secure manner"
  - Correctness
  - Privacy
  - Independence of inputs
  - Guaranteed output delivery
  - Fairness : corrupted parties should get their output iff the honest parties do
  - and ...

Multiparty Computation

- Multiparty Computation – computing a functionality f
- Secure Multiparty Computation: compute f in a “secure manner"
  - Correctness
  - Privacy
  - Independence of inputs
  - Guaranteed output delivery
  - Fairness : corrupted parties should get their output iff the honest parties do
  - and ...
- Examples: coin-tossing, broadcast, electronic voting, electronic auctions

Multiparty Computation

- Multiparty Computation – computing a functionality f
- Secure Multiparty Computation: compute f in a “secure manner"
  - Correctness
  - Privacy
  - Independence of inputs
  - Guaranteed output delivery
  - Fairness : corrupted parties should get their output iff the honest parties do
  - and ...
- Examples: coin-tossing, broadcast, electronic voting, electronic auctions
- How should we model it?

Multiparty Computation

- Multiparty Computation – computing a functionality f
- Secure Multiparty Computation: compute f in a “secure manner"
  - Correctness
  - Privacy
  - Independence of inputs
  - Guaranteed output delivery
  - Fairness : corrupted parties should get their output iff the honest parties do
  - and ...
- Examples: coin-tossing, broadcast, electronic voting, electronic auctions
- How should we model it?
- Real Vs. Ideal paradigm

Real Model Execution

For a a pair of algorithms $\overline{\mathsf{A}} = (\mathsf{A}_1, \mathsf{A}_2)$ and inputs $x_c, x_1, x_2 \in \{0,1\}^*$, let $\mathsf{REAL}_{\overline{\mathsf{A}}}(x_c, x_1, x_2)$ be the joint output of $(\mathsf{A}_1(x_c, x_1), \mathsf{A}_2(x_c, x_2))$.

Real Model Execution

For a a pair of algorithms $\overline{\mathsf{A}} = (\mathsf{A}_1, \mathsf{A}_2)$ and inputs $x_c, x_1, x_2 \in \{0,1\}^*$, let $\mathsf{REAL}_{\overline{\mathsf{A}}}(x_c, x_1, x_2)$ be the joint output of $(\mathsf{A}_1(x_c, x_1), \mathsf{A}_2(x_c, x_2))$.

Given a two-party protocol π, an algorithm taking the role of one of the parties in π is:

Real Model Execution

For a a pair of algorithms $\overline{\mathsf{A}} = (\mathsf{A}_1, \mathsf{A}_2)$ and inputs $x_c, x_1, x_2 \in \{0,1\}^*$, let $\mathsf{REAL}_{\overline{\mathsf{A}}}(x_c, x_1, x_2)$ be the joint output of $(\mathsf{A}_1(x_c, x_1), \mathsf{A}_2(x_c, x_2))$.

Given a two-party protocol π, an algorithm taking the role of one of the parties in π is:

- Malicious — acts arbitrarily.

Real Model Execution

For a a pair of algorithms $\overline{\mathsf{A}} = (\mathsf{A}_1, \mathsf{A}_2)$ and inputs $x_c, x_1, x_2 \in \{0,1\}^*$, let $\mathsf{REAL}_{\overline{\mathsf{A}}}(x_c, x_1, x_2)$ be the joint output of $(\mathsf{A}_1(x_c, x_1), \mathsf{A}_2(x_c, x_2))$.

Given a two-party protocol π, an algorithm taking the role of one of the parties in π is:

- Malicious — acts arbitrarily.
- Honest — acts exactly according to π.

Real Model Execution

For a a pair of algorithms $\overline{\mathsf{A}} = (\mathsf{A}_1, \mathsf{A}_2)$ and inputs $x_c, x_1, x_2 \in \{0,1\}^*$, let $\mathsf{REAL}_{\overline{\mathsf{A}}}(x_c, x_1, x_2)$ be the joint output of $(\mathsf{A}_1(x_c, x_1), \mathsf{A}_2(x_c, x_2))$.

Given a two-party protocol π, an algorithm taking the role of one of the parties in π is:

- Malicious — acts arbitrarily.
- Honest — acts exactly according to π.
- Semi-honest — acts according to π, but might output **additional information**.

Real Model Execution

For a a pair of algorithms $\overline{\mathsf{A}} = (\mathsf{A}_1, \mathsf{A}_2)$ and inputs $x_c, x_1, x_2 \in \{0,1\}^*$, let $\mathsf{REAL}_{\overline{\mathsf{A}}}(x_c, x_1, x_2)$ be the joint output of $(\mathsf{A}_1(x_c, x_1), \mathsf{A}_2(x_c, x_2))$.

Given a two-party protocol π, an algorithm taking the role of one of the parties in π is:

- Malicious — acts arbitrarily.
- Honest — acts exactly according to π.
- Semi-honest — acts according to π, but might output **additional information**.

Real Model Execution

For a a pair of algorithms $\overline{\mathsf{A}} = (\mathsf{A}_1, \mathsf{A}_2)$ and inputs $x_c, x_1, x_2 \in \{0,1\}^*$, let $\mathsf{REAL}_{\overline{\mathsf{A}}}(x_c, x_1, x_2)$ be the joint output of $(\mathsf{A}_1(x_c, x_1), \mathsf{A}_2(x_c, x_2))$.

Given a two-party protocol π, an algorithm taking the role of one of the parties in π is:

- Malicious — acts arbitrarily.
- Honest — acts exactly according to π.
- Semi-honest — acts according to π, but might output **additional information**.

$\overline{\mathsf{A}} = (\mathsf{A}_1, \mathsf{A}_2)$ is an admissible with respect to π, if at least one party is honest.

Ideal Model Execution

Ideal Model Execution

For a pair of oracle-aided algorithms $\overline{\mathsf{B}} = (\mathsf{B}_1, \mathsf{B}_2)$, inputs $x_c, x_1, x_2 \in \{0,1\}^*$ and a function $f = (f_1, f_2)$, let $\mathsf{IDEAL}^{f}_{\overline{\mathsf{B}}}(x_c, x_1, x_2)$ be the joint output of the parties in the end of the following experiment:

Ideal Model Execution

For a pair of oracle-aided algorithms $\overline{\mathsf{B}} = (\mathsf{B}_1, \mathsf{B}_2)$, inputs $x_c, x_1, x_2 \in \{0,1\}^*$ and a function $f = (f_1, f_2)$, let $\mathsf{IDEAL}^{f}_{\overline{\mathsf{B}}}(x_c, x_1, x_2)$ be the joint output of the parties in the end of the following experiment:

1. The input of B_i is (x_c, x_i).
2. B_i sends value y_i to the trusted party (possibly $\perp$)
3. Trusted party sends $z_i = f_i(y_0, y_1)$ to B_i (sends $\perp$, if $\perp \in \{y_0, y_1\}$)
4. Each party outputs some value.

Ideal Model Execution

For a pair of oracle-aided algorithms $\overline{\mathsf{B}} = (\mathsf{B}_1, \mathsf{B}_2)$, inputs $x_c, x_1, x_2 \in \{0,1\}^*$ and a function $f = (f_1, f_2)$, let $\mathsf{IDEAL}^f_{\overline{\mathsf{B}}}(x_c, x_1, x_2)$ be the joint output of the parties in the end of the following experiment:

1. The input of B_i is (x_c, x_i).
2. B_i sends value y_i to the trusted party (possibly $\perp$)
3. Trusted party sends $z_i = f_i(y_0, y_1)$ to B_i (sends $\perp$, if $\perp \in \{y_0, y_1\}$)
4. Each party outputs some value.

An oracle-aided algorithm B taking the role of one of the parties on the above experiment is:

Ideal Model Execution

For a pair of oracle-aided algorithms $\overline{\mathsf{B}} = (\mathsf{B}_1, \mathsf{B}_2)$, inputs $x_c, x_1, x_2 \in \{0,1\}^*$ and a function $f = (f_1, f_2)$, let $\mathrm{IDEAL}^f_{\overline{\mathsf{B}}}(x_c, x_1, x_2)$ be the joint output of the parties in the end of the following experiment:

1. The input of B_i is (x_c, x_i).
2. B_i sends value y_i to the trusted party (possibly $\perp$)
3. Trusted party sends $z_i = f_i(y_0, y_1)$ to B_i (sends $\perp$, if $\perp \in \{y_0, y_1\}$)
4. Each party outputs some value.

An oracle-aided algorithm B taking the role of one of the parties on the above experiment is:

- Malicious — acts arbitrarily.

Ideal Model Execution

For a pair of oracle-aided algorithms $\overline{\mathsf{B}} = (\mathsf{B}_1, \mathsf{B}_2)$, inputs $x_c, x_1, x_2 \in \{0,1\}^*$ and a function $f = (f_1, f_2)$, let $\mathsf{IDEAL}^{f}_{\overline{\mathsf{B}}}(x_c, x_1, x_2)$ be the joint output of the parties in the end of the following experiment:

1. The input of B_i is (x_c, x_i).
2. B_i sends value y_i to the trusted party (possibly $\perp$)
3. Trusted party sends $z_i = f_i(y_0, y_1)$ to B_i (sends $\perp$, if $\perp \in \{y_0, y_1\}$)
4. Each party outputs some value.

An oracle-aided algorithm B taking the role of one of the parties on the above experiment is:

- Malicious — acts arbitrarily.
- Honest — sends its private input to the trusted party (i.e., sets $y_i = x_i$), and its only output is the value it gets from the trusted party (i.e., z_i).

Ideal Model Execution

For a pair of oracle-aided algorithms $\overline{\mathsf{B}} = (\mathsf{B}_1, \mathsf{B}_2)$, inputs $x_c, x_1, x_2 \in \{0,1\}^*$ and a function $f = (f_1, f_2)$, let $\mathsf{IDEAL}^{f}_{\overline{\mathsf{B}}}(x_c, x_1, x_2)$ be the joint output of the parties in the end of the following experiment:

1. The input of B_i is (x_c, x_i).
2. B_i sends value y_i to the trusted party (possibly $\perp$)
3. Trusted party sends $z_i = f_i(y_0, y_1)$ to B_i (sends $\perp$, if $\perp \in \{y_0, y_1\}$)
4. Each party outputs some value.

An oracle-aided algorithm B taking the role of one of the parties on the above experiment is:

- Malicious — acts arbitrarily.
- Honest — sends its private input to the trusted party (i.e., sets $y_i = x_i$), and its only output is the value it gets from the trusted party (i.e., z_i).
- Semi-honest, sends its input to the trusted party, outputs z_i plus possibly additional information.

Ideal Model Execution

For a pair of oracle-aided algorithms $\overline{\mathsf{B}} = (\mathsf{B}_1, \mathsf{B}_2)$, inputs $x_c, x_1, x_2 \in \{0,1\}^*$ and a function $f = (f_1, f_2)$, let $\mathsf{IDEAL}^{f}_{\overline{\mathsf{B}}}(x_c, x_1, x_2)$ be the joint output of the parties in the end of the following experiment:

1. The input of B_i is (x_c, x_i).
2. B_i sends value y_i to the trusted party (possibly $\perp$)
3. Trusted party sends $z_i = f_i(y_0, y_1)$ to B_i (sends $\perp$, if $\perp \in \{y_0, y_1\}$)
4. Each party outputs some value.

An oracle-aided algorithm B taking the role of one of the parties on the above experiment is:

- Malicious — acts arbitrarily.
- Honest — sends its private input to the trusted party (i.e., sets $y_i = x_i$), and its only output is the value it gets from the trusted party (i.e., z_i).
- Semi-honest, sends its input to the trusted party, outputs z_i plus possibly additional information.

Ideal Model Execution

For a pair of oracle-aided algorithms $\overline{\mathsf{B}} = (\mathsf{B}_1, \mathsf{B}_2)$, inputs $x_c, x_1, x_2 \in \{0,1\}^*$ and a function $f = (f_1, f_2)$, let $\mathsf{IDEAL}^{f}_{\overline{\mathsf{B}}}(x_c, x_1, x_2)$ be the joint output of the parties in the end of the following experiment:

1. The input of B_i is (x_c, x_i).
2. B_i sends value y_i to the trusted party (possibly $\perp$)
3. Trusted party sends $z_i = f_i(y_0, y_1)$ to B_i (sends $\perp$, if $\perp \in \{y_0, y_1\}$)
4. Each party outputs some value.

An oracle-aided algorithm B taking the role of one of the parties on the above experiment is:

- Malicious — acts arbitrarily.
- Honest — sends its private input to the trusted party (i.e., sets $y_i = x_i$), and its only output is the value it gets from the trusted party (i.e., z_i).
- Semi-honest, sends its input to the trusted party, outputs z_i plus possibly additional information.

$\overline{\mathsf{B}} = (\mathsf{B}_1, \mathsf{B}_2)$ is admissible, if at least one party is honest.

Secure computation

Definition 1 (secure computation)

A protocol π securely computes f, if $\forall$ admissible PPT pair $\overline{\mathsf{A}} = (\mathsf{A}_1, \mathsf{A}_2)$ for π, exists admissible oracle-aided PPT pair $\overline{\mathsf{B}} = (\mathsf{B}_1, \mathsf{B}_2)$, s.t.

$$\{\mathrm{REAL}_{\overline{\mathsf{A}}}(1^n, x_1, x_2)\}_{x_c, x_1, x_2 \in \{0,1\}^*} \approx_c \{\mathrm{IDEAL}^f_{\overline{\mathsf{B}}}(x_1, x_2)\}_{x_c, x_1, x_2 \in \{0,1\}^*}$$

Secure computation

Definition 1 (secure computation)

A protocol π securely computes f, if $\forall$ admissible PPT pair $\overline{\mathsf{A}} = (\mathsf{A}_1, \mathsf{A}_2)$ for π, exists admissible oracle-aided PPT pair $\overline{\mathsf{B}} = (\mathsf{B}_1, \mathsf{B}_2)$, s.t.

$$\{\mathsf{REAL}_{\overline{\mathsf{A}}}(1^n, x_1, x_2)\}_{x_c, x_1, x_2 \in \{0,1\}^*} \approx_c \{\mathsf{IDEAL}^f_{\overline{\mathsf{B}}}(x_1, x_2)\}_{x_c, x_1, x_2 \in \{0,1\}^*}$$

- Recall that the enumeration index (i.e., x_c, x_1, x_2) is given to the distinguisher.

Secure computation

Definition 1 (secure computation)

A protocol π securely computes f, if $\forall$ admissible PPT pair $\overline{A} = (A_1, A_2)$ for π, exists admissible oracle-aided PPT pair $\overline{B} = (B_1, B_2)$, s.t.

$$\{\mathrm{REAL}_{\overline{A}}(1^n, x_1, x_2)\}_{x_c, x_1, x_2 \in \{0,1\}^*} \approx_c \{\mathrm{IDEAL}^f_{\overline{B}}(x_1, x_2)\}_{x_c, x_1, x_2 \in \{0,1\}^*}$$

- Recall that the enumeration index (i.e., x_c, x_1, x_2) is given to the distinguisher.
- π securely computes f implies that π computes f correctly.

Secure computation

Definition 1 (secure computation)

A protocol π securely computes f, if $\forall$ admissible PPT pair $\overline{\mathsf{A}} = (\mathsf{A}_1, \mathsf{A}_2)$ for π, exists admissible oracle-aided PPT pair $\overline{\mathsf{B}} = (\mathsf{B}_1, \mathsf{B}_2)$, s.t.

$$\{\mathrm{REAL}_{\overline{\mathsf{A}}}(1^n, x_1, x_2)\}_{x_c, x_1, x_2 \in \{0,1\}^*} \approx_c \{\mathrm{IDEAL}^f_{\overline{\mathsf{B}}}(x_1, x_2)\}_{x_c, x_1, x_2 \in \{0,1\}^*}$$

- Recall that the enumeration index (i.e., x_c, x_1, x_2) is given to the distinguisher.
- π securely computes f implies that π computes f correctly.
- Security parameter

Secure computation

Definition 1 (secure computation)

A protocol π securely computes f, if $\forall$ admissible PPT pair $\overline{A} = (A_1, A_2)$ for π, exists admissible oracle-aided PPT pair $\overline{B} = (B_1, B_2)$, s.t.

$$\{\mathrm{REAL}_{\overline{A}}(1^n, x_1, x_2)\}_{x_c, x_1, x_2 \in \{0,1\}^*} \approx_c \{\mathrm{IDEAL}^f_{\overline{B}}(x_1, x_2)\}_{x_c, x_1, x_2 \in \{0,1\}^*}$$

- Recall that the enumeration index (i.e., x_c, x_1, x_2) is given to the distinguisher.
- π securely computes f implies that π computes f correctly.
- Security parameter
- Auxiliary inputs

Secure computation

Definition 1 (secure computation)

A protocol π securely computes f, if $\forall$ admissible PPT pair $\overline{A} = (A_1, A_2)$ for π, exists admissible oracle-aided PPT pair $\overline{B} = (B_1, B_2)$, s.t.

$$\{\text{REAL}_{\overline{A}}(1^n, x_1, x_2)\}_{x_c, x_1, x_2 \in \{0,1\}^*} \approx_c \{\text{IDEAL}^f_{\overline{B}}(x_1, x_2)\}_{x_c, x_1, x_2 \in \{0,1\}^*}$$

- Recall that the enumeration index (i.e., x_c, x_1, x_2) is given to the distinguisher.
- π securely computes f implies that π computes f correctly.
- Security parameter
- Auxiliary inputs
- We focus on semi-honest adversaries.

Section 2

Oblivious Transfer

Oblivious transfer

An (one-out-of-two) OT protocol securely computes the functionality $\mathsf{OT} = (\mathsf{OT}_\mathsf{S}, \mathsf{OT}_\mathsf{R}))$ over $(\{0,1\}^* \times \{0,1\}^*) \times \{0,1\}$, where $\mathsf{OT}_\mathsf{S}(\cdot) = \perp$ and $\mathsf{OT}_\mathsf{R}((\sigma_0, \sigma_1), i) = \sigma_i$.

Oblivious transfer

An (one-out-of-two) OT protocol securely computes the functionality $\mathsf{OT} = (\mathsf{OT}_\mathsf{S}, \mathsf{OT}_\mathsf{R}))$ over $(\{0,1\}^* \times \{0,1\}^*) \times \{0,1\}$, where $\mathsf{OT}_\mathsf{S}(\cdot) = \perp$ and $\mathsf{OT}_\mathsf{R}((\sigma_0, \sigma_1), i) = \sigma_i$.

- “Complete" for multiparty computation

Oblivious transfer

An (one-out-of-two) OT protocol securely computes the functionality $\mathsf{OT} = (\mathsf{OT}_\mathsf{S}, \mathsf{OT}_\mathsf{R}))$ over $(\{0,1\}^* \times \{0,1\}^*) \times \{0,1\}$, where $\mathsf{OT}_\mathsf{S}(\cdot) = \perp$ and $\mathsf{OT}_\mathsf{R}((\sigma_0, \sigma_1), i) = \sigma_i$.

- “Complete" for multiparty computation
- We show how to construct for bit inputs.

Oblivious transfer from trapdoor permutations

Let (G, f, Inv) be a TDP and let b be an hardcore predicate for f.

Oblivious transfer from trapdoor permutations

Let (G, f, Inv) be a TDP and let b be an hardcore predicate for f.

Protocol 2 ((S, R))

Common input: 1^n
S**'s input:** $\sigma_0, \sigma_1 \in \{0, 1\}$.
R**'s input:** $i \in \{0, 1\}$.

1. S chooses $(e, d) \leftarrow \mathsf{G}(1^n)$, and sends e to R.
2. R chooses $x_0, x_1 \leftarrow \{0, 1\}^n$, sets $y_i = f_e(x_i)$ and $y_{1-i} = x_{1-i}$, and sends y_0, y_1 to S.
3. S sets $c_j = b(\mathrm{Inv}_d(y_j)) \oplus \sigma_j$, for $j \in \{0, 1\}$, and sends (c_0, c_1) to R.
4. R outputs $c_i \oplus b(x_i)$.

Oblivious transfer from trapdoor permutations

Let (G, f, Inv) be a TDP and let b be an hardcore predicate for f.

Protocol 2 ((S, R))

Common input: 1^n
S**'s input:** $\sigma_0, \sigma_1 \in \{0, 1\}$.
R**'s input:** $i \in \{0, 1\}$.

1. S chooses $(e, d) \leftarrow \mathsf{G}(1^n)$, and sends e to R.
2. R chooses $x_0, x_1 \leftarrow \{0, 1\}^n$, sets $y_i = f_e(x_i)$ and $y_{1-i} = x_{1-i}$, and sends y_0, y_1 to S.
3. S sets $c_j = b(\mathsf{Inv}_d(y_j)) \oplus \sigma_j$, for $j \in \{0, 1\}$, and sends (c_0, c_1) to R.
4. R outputs $c_i \oplus b(x_i)$.

Claim 3

Protocol 2 securely computes OT (in the semi-honest model).

Proving Claim 3

We need to prove that $\forall$ semi-honest admissible PPT pair $\overline{\mathsf{A}} = (\mathsf{A}_1, \mathsf{A}_2)$ for (S, R), exists admissible oracle-aided PPT pair $\overline{\mathsf{B}} = (\mathsf{B}_1, \mathsf{B}_2)$ s.t.

$$\{\mathsf{REAL}_{\overline{\mathsf{A}}}(1^n, (\sigma_0, \sigma_1), i)\} \approx_c \{\mathsf{IDEAL}^{\mathsf{OT}}_{\overline{\mathsf{B}}}(1^n, (\sigma_0, \sigma_1), i)\}, \quad (1)$$

where the enumeration is over $n \in \mathbb{N}$ and $\sigma_0, \sigma_1, i \in \{0, 1\}$.

R's privacy

For a semi-honest implementation S' of S, define the oracle-aided semi-honest strategy $\mathsf{S}'_{\mathcal{I}}$ as follows.

R's privacy

For a semi-honest implementation S' of S, define the oracle-aided semi-honest strategy $\mathsf{S}'_{\mathcal{I}}$ as follows.

Algorithm 4 ($\mathsf{S}'_{\mathcal{I}}$)

input: $1^n, \sigma_0, \sigma_1$

1. Send (σ_0, σ_1) to the trusted party.
2. Emulate $(\mathsf{S}'(1^n, \sigma_0, \sigma_1), \mathsf{R}(1^n, 0))$.
3. Output the output that S' does.

R's privacy

For a semi-honest implementation S' of S, define the oracle-aided semi-honest strategy $\mathsf{S}'_{\mathcal{I}}$ as follows.

Algorithm 4 ($\mathsf{S}'_{\mathcal{I}}$)

input: $1^n, \sigma_0, \sigma_1$

1. Send (σ_0, σ_1) to the trusted party.
2. Emulate $(\mathsf{S}'(1^n, \sigma_0, \sigma_1), \mathsf{R}(1^n, 0))$.
3. Output the output that S' does.

Let $\overline{\mathsf{A}} = (\mathsf{S}', \mathsf{R})$ and $\overline{\mathsf{B}} = (\mathsf{S}'_{\mathcal{I}}, \mathsf{R}_{\mathcal{I}})$, where $\mathsf{R}_{\mathcal{I}}$ is honest.

R's privacy

For a semi-honest implementation S' of S, define the oracle-aided semi-honest strategy $\mathsf{S}'_{\mathcal{I}}$ as follows.

Algorithm 4 ($\mathsf{S}'_{\mathcal{I}}$)

input: $1^n, \sigma_0, \sigma_1$

1. Send (σ_0, σ_1) to the trusted party.
2. Emulate $(\mathsf{S}'(1^n, \sigma_0, \sigma_1), \mathsf{R}(1^n, 0))$.
3. Output the output that S' does.

Let $\overline{\mathsf{A}} = (\mathsf{S}', \mathsf{R})$ and $\overline{\mathsf{B}} = (\mathsf{S}'_{\mathcal{I}}, \mathsf{R}_{\mathcal{I}})$, where $\mathsf{R}_{\mathcal{I}}$ is honest.

Claim 5

$\{\mathsf{REAL}_{\overline{\mathsf{A}}}(1^n, (\sigma_0, \sigma_1), i)\} \equiv \{\mathsf{IDEAL}^{\mathsf{OT}}_{\overline{\mathsf{B}}}(1^n, (\sigma_0, \sigma_1), i)\}$.

R's privacy

For a semi-honest implementation S' of S, define the oracle-aided semi-honest strategy $\mathsf{S}'_{\mathcal{I}}$ as follows.

Algorithm 4 ($\mathsf{S}'_{\mathcal{I}}$)

input: $1^n, \sigma_0, \sigma_1$

1. Send (σ_0, σ_1) to the trusted party.
2. Emulate $(\mathsf{S}'(1^n, \sigma_0, \sigma_1), \mathsf{R}(1^n, 0))$.
3. Output the output that S' does.

Let $\overline{\mathsf{A}} = (\mathsf{S}', \mathsf{R})$ and $\overline{\mathsf{B}} = (\mathsf{S}'_{\mathcal{I}}, \mathsf{R}_{\mathcal{I}})$, where $\mathsf{R}_{\mathcal{I}}$ is honest.

Claim 5

$\{\mathsf{REAL}_{\overline{\mathsf{A}}}(1^n, (\sigma_0, \sigma_1), i)\} \equiv \{\mathsf{IDEAL}^{\mathsf{OT}}_{\overline{\mathsf{B}}}(1^n, (\sigma_0, \sigma_1), i)\}$.

Proof?

S's privacy

For a semi-honest implementation R' of R, define the oracle-aided semi-honest strategy $\mathsf{R}'_{\mathcal{I}}$ as follows.

S's privacy

For a semi-honest implementation R' of R, define the oracle-aided semi-honest strategy $\mathsf{R}'_{\mathcal{I}}$ as follows.

Algorithm 6 ($\mathsf{R}'_{\mathcal{I}}$)

input: $1^n, i \in \{0,1\}$,

1. Send i to the trusted party, and let σ be its answer.
2. Emulate $(\mathsf{S}(1^n, \sigma_0, \sigma_1), \mathsf{R}'(1^n, i))$, for $\sigma_i = \sigma$ and $\sigma_{1-i} = 0$.
3. Output the output that R' does.

S's privacy

For a semi-honest implementation R' of R, define the oracle-aided semi-honest strategy $\mathsf{R}'_{\mathcal{I}}$ as follows.

Algorithm 6 ($\mathsf{R}'_{\mathcal{I}}$)

input: $1^n, i \in \{0,1\}$,

1. Send i to the trusted party, and let σ be its answer.
2. Emulate $(\mathsf{S}(1^n, \sigma_0, \sigma_1), \mathsf{R}'(1^n, i))$, for $\sigma_i = \sigma$ and $\sigma_{1-i} = 0$.
3. Output the output that R' does.

Let $\overline{\mathsf{A}} = (\mathsf{S}, \mathsf{R}')$ and $\overline{\mathsf{B}} = (\mathsf{S}_{\mathcal{I}}, \mathsf{R}'_{\mathcal{I}})$, where $\mathsf{S}_{\mathcal{I}}$ is honest.

S's privacy

For a semi-honest implementation R' of R, define the oracle-aided semi-honest strategy $\mathsf{R}'_{\mathcal{I}}$ as follows.

Algorithm 6 ($\mathsf{R}'_{\mathcal{I}}$)

input: $1^n, i \in \{0,1\}$,

1. Send i to the trusted party, and let σ be its answer.
2. Emulate $(\mathsf{S}(1^n, \sigma_0, \sigma_1), \mathsf{R}'(1^n, i))$, for $\sigma_i = \sigma$ and $\sigma_{1-i} = 0$.
3. Output the output that R' does.

Let $\overline{\mathsf{A}} = (\mathsf{S}, \mathsf{R}')$ and $\overline{\mathsf{B}} = (\mathsf{S}_{\mathcal{I}}, \mathsf{R}'_{\mathcal{I}})$, where $\mathsf{S}_{\mathcal{I}}$ is honest.

Claim 7

$\{\mathsf{REAL}_{\overline{\mathsf{A}}}(1^n, (\sigma_0, \sigma_1), i)\} \approx_c \{\mathsf{IDEAL}^{\mathsf{OT}}_{\overline{\mathsf{B}}}(1^n, (\sigma_0, \sigma_1), i)\}$.

S's privacy

For a semi-honest implementation R' of R, define the oracle-aided semi-honest strategy $\mathsf{R}'_{\mathcal{I}}$ as follows.

Algorithm 6 ($\mathsf{R}'_{\mathcal{I}}$)

input: $1^n, i \in \{0,1\}$,

1. Send i to the trusted party, and let σ be its answer.
2. Emulate $(\mathsf{S}(1^n, \sigma_0, \sigma_1), \mathsf{R}'(1^n, i))$, for $\sigma_i = \sigma$ and $\sigma_{1-i} = 0$.
3. Output the output that R' does.

Let $\overline{\mathsf{A}} = (\mathsf{S}, \mathsf{R}')$ and $\overline{\mathsf{B}} = (\mathsf{S}_{\mathcal{I}}, \mathsf{R}'_{\mathcal{I}})$, where $\mathsf{S}_{\mathcal{I}}$ is honest.

Claim 7

$\{\mathrm{REAL}_{\overline{\mathsf{A}}}(1^n, (\sigma_0, \sigma_1), i)\} \approx_c \{\mathrm{IDEAL}^{\mathrm{OT}}_{\overline{\mathsf{B}}}(1^n, (\sigma_0, \sigma_1), i)\}$.

Proof?

Section 3

Yao Garbled Circuit

Before we start

- Fix a (multiple message) semantically-secure private-key encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ with
  1. $\mathsf{G}(1^n) = U_n$.
  2. For any $m \in \{0,1\}^*$
     $\Pr_{d,d' \leftarrow \{0,1\}^n}[\mathsf{D}_d(\mathsf{E}_{d'}(m)) \neq \perp] = \mathsf{neg}(n)$.

Before we start

- Fix a (multiple message) semantically-secure private-key encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ with
  1. $\mathsf{G}(1^n) = U_n$.
  2. For any $m \in \{0,1\}^*$
     $\Pr_{d,d' \leftarrow \{0,1\}^n}[\mathsf{D}_d(\mathsf{E}_{d'}(m)) \neq \perp] = \operatorname{neg}(n)$.

  Can we construct such a scheme?

Before we start

- Fix a (multiple message) semantically-secure private-key encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ with
  1. $\mathsf{G}(1^n) = U_n$.
  2. For any $m \in \{0,1\}^*$
     $\Pr_{d,d' \leftarrow \{0,1\}^n}[\mathsf{D}_d(\mathsf{E}_{d'}(m)) \neq \perp] = \mathsf{neg}(n)$.

  Can we construct such a scheme?
- Boolean circuits: gates, wires, inputs, outputs, values, computation

The Garbled Circuit

The Garbled Circuit

Fix a Boolean circuit C and $n \in \mathbb{N}$.

The Garbled Circuit

Fix a Boolean circuit C and $n \in \mathbb{N}$.

- Let $\mathcal{W}$ and $\mathcal{G}$ be the (indices) of **wires** and **gates** of C, respectively.

The Garbled Circuit

Fix a Boolean circuit C and $n \in \mathbb{N}$.

- Let $\mathcal{W}$ and $\mathcal{G}$ be the (indices) of **wires** and **gates** of C, respectively.
- For $w \in \mathcal{W}$, associate a pair of random 'keys" $k_w = (k_w^0, k_w^1) \in (\{0,1\}^n)^2$.

The Garbled Circuit

Fix a Boolean circuit C and $n \in \mathbb{N}$.

- Let $\mathcal{W}$ and $\mathcal{G}$ be the (indices) of **wires** and **gates** of C, respectively.
- For $w \in \mathcal{W}$, associate a pair of random ‘keys" $k_w = (k_w^0, k_w^1) \in (\{0,1\}^n)^2$.
- For $g \in \mathcal{G}$ with input wires i and j, and output wire h, let $T(g)$ be the following table:

The Garbled Circuit

Fix a Boolean circuit C and $n \in \mathbb{N}$.

- Let $\mathcal{W}$ and $\mathcal{G}$ be the (indices) of **wires** and **gates** of C, respectively.
- For $w \in \mathcal{W}$, associate a pair of random 'keys" $k_w = (k_w^0, k_w^1) \in (\{0,1\}^n)^2$.
- For $g \in \mathcal{G}$ with input wires i and j, and output wire h, let $T(g)$ be the following table:

The Garbled Circuit

Fix a Boolean circuit C and $n \in \mathbb{N}$.

- Let $\mathcal{W}$ and $\mathcal{G}$ be the (indices) of **wires** and **gates** of C, respectively.
- For $w \in \mathcal{W}$, associate a pair of random ‘keys" $k_w = (k_w^0, k_w^1) \in (\{0,1\}^n)^2$.
- For $g \in \mathcal{G}$ with input wires i and j, and output wire h, let $T(g)$ be the following table:

input wire i	input wire j	output wire h	hidden output wire
k_i^0	k_j^0	$k_h^{g(0,0)}$	$E_{k_i^0}(E_{k_j^0}(k_h^{g(0,0)}))$
k_i^0	k_j^1	$k_h^{g(0,1)}$	$E_{k_i^0}(E_{k_j^1}(k_h^{g(0,1)}))$
k_i^1	k_j^0	$k_h^{g(1,0)}$	$E_{k_i^1}(E_{k_j^0}(k_h^{g(1,0)}))$
k_i^1	k_j^1	$k_h^{g(1,1)}$	$E_{k_i^1}(E_{k_j^1}(k_h^{g(1,1)}))$

Figure: Table for gate g, with input wires i and j, and output wire h.

The Garbled Circuit, cont.

input wire i	input wire j	output wire h	hidden output wire
k_i^0	k_j^0	$k_h^{g(0,0)}$	$E_{k_i^0}(E_{k_j^0}(k_h^{g(0,0)}))$
k_i^0	k_j^1	$k_h^{g(0,1)}$	$E_{k_i^0}(E_{k_j^1}(k_h^{g(0,1)}))$
k_i^1	k_j^0	$k_h^{g(1,0)}$	$E_{k_i^1}(E_{k_j^0}(k_h^{g(1,0)}))$
k_i^1	k_j^1	$k_h^{g(1,1)}$	$E_{k_i^1}(E_{k_j^1}(k_h^{g(1,1)}))$

The Garbled Circuit, cont.

input wire i	input wire j	output wire h	hidden output wire
k_i^0	k_j^0	$k_h^{g(0,0)}$	$E_{k_i^0}(E_{k_j^0}(k_h^{g(0,0)}))$
k_i^0	k_j^1	$k_h^{g(0,1)}$	$E_{k_i^0}(E_{k_j^1}(k_h^{g(0,1)}))$
k_i^1	k_j^0	$k_h^{g(1,0)}$	$E_{k_i^1}(E_{k_j^0}(k_h^{g(1,0)}))$
k_i^1	k_j^1	$k_h^{g(1,1)}$	$E_{k_i^1}(E_{k_j^1}(k_h^{g(1,1)}))$

Let $\mathcal{I}$ and $\mathcal{O}$ be the input and outputs wires of C.

The Garbled Circuit, cont.

input wire i	input wire j	output wire h	hidden output wire
k_i^0	k_j^0	$k_h^{g(0,0)}$	$E_{k_i^0}(E_{k_j^0}(k_h^{g(0,0)}))$
k_i^0	k_j^1	$k_h^{g(0,1)}$	$E_{k_i^0}(E_{k_j^1}(k_h^{g(0,1)}))$
k_i^1	k_j^0	$k_h^{g(1,0)}$	$E_{k_i^1}(E_{k_j^0}(k_h^{g(1,0)}))$
k_i^1	k_j^1	$k_h^{g(1,1)}$	$E_{k_i^1}(E_{k_j^1}(k_h^{g(1,1)}))$

Let $\mathcal{I}$ and $\mathcal{O}$ be the input and outputs wires of C.

- For $g \in \mathcal{G}$, let $\widetilde{T}(g)$ be a random permutation of the fourth column of $T(g)$.

The Garbled Circuit, cont.

input wire i	input wire j	output wire h	hidden output wire
k_i^0	k_j^0	$k_h^{g(0,0)}$	$E_{k_i^0}(E_{k_j^0}(k_h^{g(0,0)}))$
k_i^0	k_j^1	$k_h^{g(0,1)}$	$E_{k_i^0}(E_{k_j^1}(k_h^{g(0,1)}))$
k_i^1	k_j^0	$k_h^{g(1,0)}$	$E_{k_i^1}(E_{k_j^0}(k_h^{g(1,0)}))$
k_i^1	k_j^1	$k_h^{g(1,1)}$	$E_{k_i^1}(E_{k_j^1}(k_h^{g(1,1)}))$

Let $\mathcal{I}$ and $\mathcal{O}$ be the input and outputs wires of C.

- For $g \in \mathcal{G}$, let $\widetilde{T}(g)$ be a random permutation of the fourth column of $T(g)$.
- Let $C(x)_w$ be the **bit-value** the computation of $C(x)$ assigns to w.

The Garbled Circuit, cont.

input wire i	input wire j	output wire h	hidden output wire
k_i^0	k_j^0	$k_h^{g(0,0)}$	$E_{k_i^0}(E_{k_j^0}(k_h^{g(0,0)}))$
k_i^0	k_j^1	$k_h^{g(0,1)}$	$E_{k_i^0}(E_{k_j^1}(k_h^{g(0,1)}))$
k_i^1	k_j^0	$k_h^{g(1,0)}$	$E_{k_i^1}(E_{k_j^0}(k_h^{g(1,0)}))$
k_i^1	k_j^1	$k_h^{g(1,1)}$	$E_{k_i^1}(E_{k_j^1}(k_h^{g(1,1)}))$

Let $\mathcal{I}$ and $\mathcal{O}$ be the input and outputs wires of C.

- For $g \in \mathcal{G}$, let $\widetilde{T}(g)$ be a random permutation of the fourth column of $T(g)$.
- Let $C(x)_w$ be the **bit-value** the computation of $C(x)$ assigns to w.
- Given

The Garbled Circuit, cont.

input wire i	input wire j	output wire h	hidden output wire
k_i^0	k_j^0	$k_h^{g(0,0)}$	$E_{k_i^0}(E_{k_j^0}(k_h^{g(0,0)}))$
k_i^0	k_j^1	$k_h^{g(0,1)}$	$E_{k_i^0}(E_{k_j^1}(k_h^{g(0,1)}))$
k_i^1	k_j^0	$k_h^{g(1,0)}$	$E_{k_i^1}(E_{k_j^0}(k_h^{g(1,0)}))$
k_i^1	k_j^1	$k_h^{g(1,1)}$	$E_{k_i^1}(E_{k_j^1}(k_h^{g(1,1)}))$

Let $\mathcal{I}$ and $\mathcal{O}$ be the input and outputs wires of C.

- For $g \in \mathcal{G}$, let $\widetilde{T}(g)$ be a random permutation of the fourth column of $T(g)$.
- Let $C(x)_w$ be the **bit-value** the computation of $C(x)$ assigns to w.
- Given
  1. $\widetilde{T} = \{(g, \widetilde{T}(g))\}_{g \in \mathcal{G}}$.

The Garbled Circuit, cont.

input wire i	input wire j	output wire h	hidden output wire
k_i^0	k_j^0	$k_h^{g(0,0)}$	$E_{k_i^0}(E_{k_j^0}(k_h^{g(0,0)}))$
k_i^0	k_j^1	$k_h^{g(0,1)}$	$E_{k_i^0}(E_{k_j^1}(k_h^{g(0,1)}))$
k_i^1	k_j^0	$k_h^{g(1,0)}$	$E_{k_i^1}(E_{k_j^0}(k_h^{g(1,0)}))$
k_i^1	k_j^1	$k_h^{g(1,1)}$	$E_{k_i^1}(E_{k_j^1}(k_h^{g(1,1)}))$

Let $\mathcal{I}$ and $\mathcal{O}$ be the input and outputs wires of C.

- For $g \in \mathcal{G}$, let $\widetilde{T}(g)$ be a random permutation of the fourth column of $T(g)$.
- Let $C(x)_w$ be the **bit-value** the computation of $C(x)$ assigns to w.
- Given
  1. $\widetilde{T} = \{(g, \widetilde{T}(g))\}_{g \in \mathcal{G}}$.
  2. $\{k_w^{C(x)_w}\}_{w \in \mathcal{I}}$ for some x.

The Garbled Circuit, cont.

input wire i	input wire j	output wire h	hidden output wire
k_i^0	k_j^0	$k_h^{g(0,0)}$	$E_{k_i^0}(E_{k_j^0}(k_h^{g(0,0)}))$
k_i^0	k_j^1	$k_h^{g(0,1)}$	$E_{k_i^0}(E_{k_j^1}(k_h^{g(0,1)}))$
k_i^1	k_j^0	$k_h^{g(1,0)}$	$E_{k_i^1}(E_{k_j^0}(k_h^{g(1,0)}))$
k_i^1	k_j^1	$k_h^{g(1,1)}$	$E_{k_i^1}(E_{k_j^1}(k_h^{g(1,1)}))$

Let $\mathcal{I}$ and $\mathcal{O}$ be the input and outputs wires of C.

- For $g \in \mathcal{G}$, let $\widetilde{T}(g)$ be a random permutation of the fourth column of $T(g)$.
- Let $C(x)_w$ be the **bit-value** the computation of $C(x)$ assigns to w.
- Given
  1. $\widetilde{T} = \{(g, \widetilde{T}(g))\}_{g \in \mathcal{G}}$.
  2. $\{k_w^{C(x)_w}\}_{w \in \mathcal{I}}$ for some x.
  3. $\{(w, k_w = (k_w^0, k_w^1)\}_{w \in \mathcal{O}}$.

The Garbled Circuit, cont.

input wire i	input wire j	output wire h	hidden output wire
k_i^0	k_j^0	$k_h^{g(0,0)}$	$E_{k_i^0}(E_{k_j^0}(k_h^{g(0,0)}))$
k_i^0	k_j^1	$k_h^{g(0,1)}$	$E_{k_i^0}(E_{k_j^1}(k_h^{g(0,1)}))$
k_i^1	k_j^0	$k_h^{g(1,0)}$	$E_{k_i^1}(E_{k_j^0}(k_h^{g(1,0)}))$
k_i^1	k_j^1	$k_h^{g(1,1)}$	$E_{k_i^1}(E_{k_j^1}(k_h^{g(1,1)}))$

Let $\mathcal{I}$ and $\mathcal{O}$ be the input and outputs wires of C.

- For $g \in \mathcal{G}$, let $\widetilde{T}(g)$ be a random permutation of the fourth column of $T(g)$.
- Let $C(x)_w$ be the **bit-value** the computation of $C(x)$ assigns to w.
- Given
  1. $\widetilde{T} = \{(g, \widetilde{T}(g))\}_{g \in \mathcal{G}}$.
  2. $\{k_w^{C(x)_w}\}_{w \in \mathcal{I}}$ for some x.
  3. $\{(w, k_w = (k_w^0, k_w^1)\}_{w \in \mathcal{O}}$.

The Garbled Circuit, cont.

input wire i	input wire j	output wire h	hidden output wire
k_i^0	k_j^0	$k_h^{g(0,0)}$	$E_{k_i^0}(E_{k_j^0}(k_h^{g(0,0)}))$
k_i^0	k_j^1	$k_h^{g(0,1)}$	$E_{k_i^0}(E_{k_j^1}(k_h^{g(0,1)}))$
k_i^1	k_j^0	$k_h^{g(1,0)}$	$E_{k_i^1}(E_{k_j^0}(k_h^{g(1,0)}))$
k_i^1	k_j^1	$k_h^{g(1,1)}$	$E_{k_i^1}(E_{k_j^1}(k_h^{g(1,1)}))$

Let $\mathcal{I}$ and $\mathcal{O}$ be the input and outputs wires of C.

- For $g \in \mathcal{G}$, let $\widetilde{T}(g)$ be a random permutation of the fourth column of $T(g)$.
- Let $C(x)_w$ be the **bit-value** the computation of $C(x)$ assigns to w.
- Given
  1. $\widetilde{T} = \{(g, \widetilde{T}(g))\}_{g \in \mathcal{G}}$.
  2. $\{k_w^{C(x)_w}\}_{w \in \mathcal{I}}$ for some x.
  3. $\{(w, k_w = (k_w^0, k_w^1))\}_{w \in \mathcal{O}}$.

  One can efficiently compute $C(x)$.

The Garbled Circuit, cont.

input wire i	input wire j	output wire h	hidden output wire
k_i^0	k_j^0	$k_h^{g(0,0)}$	$E_{k_i^0}(E_{k_j^0}(k_h^{g(0,0)}))$
k_i^0	k_j^1	$k_h^{g(0,1)}$	$E_{k_i^0}(E_{k_j^1}(k_h^{g(0,1)}))$
k_i^1	k_j^0	$k_h^{g(1,0)}$	$E_{k_i^1}(E_{k_j^0}(k_h^{g(1,0)}))$
k_i^1	k_j^1	$k_h^{g(1,1)}$	$E_{k_i^1}(E_{k_j^1}(k_h^{g(1,1)}))$

Let $\mathcal{I}$ and $\mathcal{O}$ be the input and outputs wires of C.

- For $g \in \mathcal{G}$, let $\widetilde{T}(g)$ be a random permutation of the fourth column of $T(g)$.
- Let $C(x)_w$ be the **bit-value** the computation of $C(x)$ assigns to w.
- Given
  1. $\widetilde{T} = \{(g, \widetilde{T}(g))\}_{g \in \mathcal{G}}$.
  2. $\{k_w^{C(x)_w}\}_{w \in \mathcal{I}}$ for some x.
  3. $\{(w, k_w = (k_w^0, k_w^1)\}_{w \in \mathcal{O}}$.

  One can efficiently compute $C(x)$.
- (essentially) No additional information about x leaks.

The Protocol

- Let $f(x_A, x_B) = (f_A(x_A, x_B), f_B(x_A, x_B))$ be function and let C be a circuit that computes f.

The Protocol

- Let $f(x_A, x_B) = (f_A(x_A, x_B), f_B(x_A, x_B))$ be function and let C be a circuit that computes f.
- Let $\mathcal{I}_A$ and $\mathcal{I}_B$ be the input wires corresponds to x_A and x_B respectively in C, and let $\mathcal{O}_A$ and $\mathcal{O}_B$ be the output wires corresponds to f_A and f_B outputs respectively in C.

The Protocol

- Let $f(x_A, x_B) = (f_A(x_A, x_B), f_B(x_A, x_B))$ be function and let C be a circuit that computes f.
- Let $\mathcal{I}_A$ and $\mathcal{I}_B$ be the input wires corresponds to x_A and x_B respectively in C, and let $\mathcal{O}_A$ and $\mathcal{O}_B$ be the output wires corresponds to f_A and f_B outputs respectively in C.
- Recall that $C(x)_w$ is the bit-value the computation of $C(x)$ assigns to w.

The Protocol

- Let $f(x_A, x_B) = (f_A(x_A, x_B), f_B(x_A, x_B))$ be function and let C be a circuit that computes f.
- Let $\mathcal{I}_A$ and $\mathcal{I}_B$ be the input wires corresponds to x_A and x_B respectively in C, and let $\mathcal{O}_A$ and $\mathcal{O}_B$ be the output wires corresponds to f_A and f_B outputs respectively in C.
- Recall that $C(x)_w$ is the bit-value the computation of $C(x)$ assigns to w.
- Let (S, R) be a secure protocol for OT.

The Protocol

- Let $f(x_A, x_B) = (f_A(x_A, x_B), f_B(x_A, x_B))$ be function and let C be a circuit that computes f.
- Let $\mathcal{I}_A$ and $\mathcal{I}_B$ be the input wires corresponds to x_A and x_B respectively in C, and let $\mathcal{O}_A$ and $\mathcal{O}_B$ be the output wires corresponds to f_A and f_B outputs respectively in C.
- Recall that $C(x)_w$ is the bit-value the computation of $C(x)$ assigns to w.
- Let (S, R) be a secure protocol for OT.

The Protocol

- Let $f(x_A, x_B) = (f_A(x_A, x_B), f_B(x_A, x_B))$ be function and let C be a circuit that computes f.
- Let $\mathcal{I}_A$ and $\mathcal{I}_B$ be the input wires corresponds to x_A and x_B respectively in C, and let $\mathcal{O}_A$ and $\mathcal{O}_B$ be the output wires corresponds to f_A and f_B outputs respectively in C.
- Recall that $C(x)_w$ is the bit-value the computation of $C(x)$ assigns to w.
- Let (S, R) be a secure protocol for OT.

Protocol 8 ((A, B))

Common input: 1^n. A/B**'s input:** x_A / x_B

1. A samples at random $\{k_w = (k_w^0, k_w^1)\}_{w \in \mathcal{W}}$, and generate $\widetilde{T}$).
2. A sends $\widetilde{T}$ and $\{(w, k_w^{C(x_1,\cdot)_w})\}_{w \in \mathcal{I}_A}$ to B.
3. $\forall w \in \mathcal{I}_B$, A and B interact in $(S(k_w), R(C(\cdot, x_2)_w))(1^n)$.
4. B computes the (garbled) circuit, and sends $\{(w, k_w^{C(x_1,x_2)_w})\}_{w \in \mathcal{O}_A}$ to A.
5. A sends $\{(w, k_w)\}_{w \in \mathcal{O}_B}$ to B.
6. The parties compute $f_A(x_1, x_2)$ and $f_B(x_1, x_2)$ respectively.

Claim 9

Protocol 8 securely computes f (in the semi-honest model)

Claim 9

Protocol 8 securely computes f (in the semi-honest model)

Proof:

Claim 9

Protocol 8 securely computes f (in the semi-honest model)

Proof: We focus on A's privacy.

Claim 9

Protocol 8 securely computes f (in the semi-honest model)

Proof: We focus on A's privacy. For a semi-honest B′, define

Algorithm 10 ($\mathsf{B}'_{\mathcal{I}}$)

input: 1^n and x_{B}.

1. Send x_{B} to the trusted party, and let o_{B} be its answer.
2. Emulate the first 4 steps of $(\mathsf{A}(1^{|x_{\mathsf{A}}|}), \mathsf{B}'(x_{\mathsf{B}})(1^n))$.
3. For each $w \in \mathcal{O}_{\mathsf{B}}$: permute the order of the pair k_w according to o_{B}, and the key of w computed in the emulation.
4. Complete the emulation, and output the output that B′ does.

Claim 9

Protocol 8 securely computes f (in the semi-honest model)

Proof: We focus on A's privacy. For a semi-honest B', define

Algorithm 10 ($\mathsf{B}'_{\mathcal{I}}$)

input: 1^n and x_{B}.

1. Send x_{B} to the trusted party, and let o_{B} be its answer.
2. Emulate the first 4 steps of $(\mathsf{A}(1^{|x_{\mathsf{A}}|}), \mathsf{B}'(x_{\mathsf{B}})(1^n))$.
3. For each $w \in \mathcal{O}_{\mathsf{B}}$: permute the order of the pair k_w according to o_{B}, and the key of w computed in the emulation.
4. Complete the emulation, and output the output that B' does.

Claim: $\mathsf{B}'_{\mathcal{I}}$ is a good "simulator" for B'.

Extensions

Extensions

- Efficiently computable f

  Both parties first compute C_f – a circuit that compute f for inputs of the right length

Extensions

- Efficiently computable f

  Both parties first compute C_f – a circuit that compute f for inputs of the right length
- Hiding C?

Extensions

- Efficiently computable f

  Both parties first compute C_f – a circuit that compute f for inputs of the right length

- Hiding C?

Extensions

- Efficiently computable f

  Both parties first compute C_f – a circuit that compute f for inputs of the right length
- Hiding C? All but its size

Malicious model

The parties prove that they act “honestly"

Malicious model

The parties prove that they act “honestly"

1. Forces the parties to chose their random coin properly

Malicious model

The parties prove that they act “honestly"

1. Forces the parties to chose their random coin properly
2. Before each step, the parties prove in $\mathcal{ZK}$ that they followed the prescribed protocol (with respect to the random-coins chosen above)

Malicious model

The parties prove that they act “honestly"

1. Forces the parties to chose their random coin properly
2. Before each step, the parties prove in $\mathcal{ZK}$ that they followed the prescribed protocol (with respect to the random-coins chosen above)

Malicious model

The parties prove that they act “honestly"

1. Forces the parties to chose their random coin properly
2. Before each step, the parties prove in $\mathcal{ZK}$ that they followed the prescribed protocol (with respect to the random-coins chosen above)

More efficient alternatives: “cut and choose"

Course Summary

See diagram

What we did not cover

- “Few" reductions

What we did not cover

- “Few" reductions
- Environment security (e.g., UC)

What we did not cover

- “Few" reductions
- Environment security (e.g., UC)
- Information theoretic crypto

What we did not cover

- “Few" reductions
- Environment security (e.g., UC)
- Information theoretic crypto
- Non-generic constructions : number theory, lattices

What we did not cover

- “Few" reductions
- Environment security (e.g., UC)
- Information theoretic crypto
- Non-generic constructions : number theory, lattices
- Impossibility results

What we did not cover

- “Few" reductions
- Environment security (e.g., UC)
- Information theoretic crypto
- Non-generic constructions : number theory, lattices
- Impossibility results
- “Real life cryptography" (e.g., Random oracle model)

What we did not cover

- “Few" reductions
- Environment security (e.g., UC)
- Information theoretic crypto
- Non-generic constructions : number theory, lattices
- Impossibility results
- “Real life cryptography" (e.g., Random oracle model)
- Security

What we did not cover

- “Few" reductions
- Environment security (e.g., UC)
- Information theoretic crypto
- Non-generic constructions : number theory, lattices
- Impossibility results
- “Real life cryptography" (e.g., Random oracle model)
- Security
- Differential Privacy

What we did not cover

- “Few" reductions
- Environment security (e.g., UC)
- Information theoretic crypto
- Non-generic constructions : number theory, lattices
- Impossibility results
- “Real life cryptography" (e.g., Random oracle model)
- Security
- Differential Privacy
- and....